Vertex Programming With OpenGL

Session 205

Vertex Programming With OpenGL

Michael K. Larson
Graphics Guy
mklarson@apple.com

Introduction

- OpenGL vertex programs
  - Vertex programs in the OpenGL pipeline
  - Vertex unit computation model
  - ARB vertex program syntax
  - Program examples

Why Use Vertex Programs?

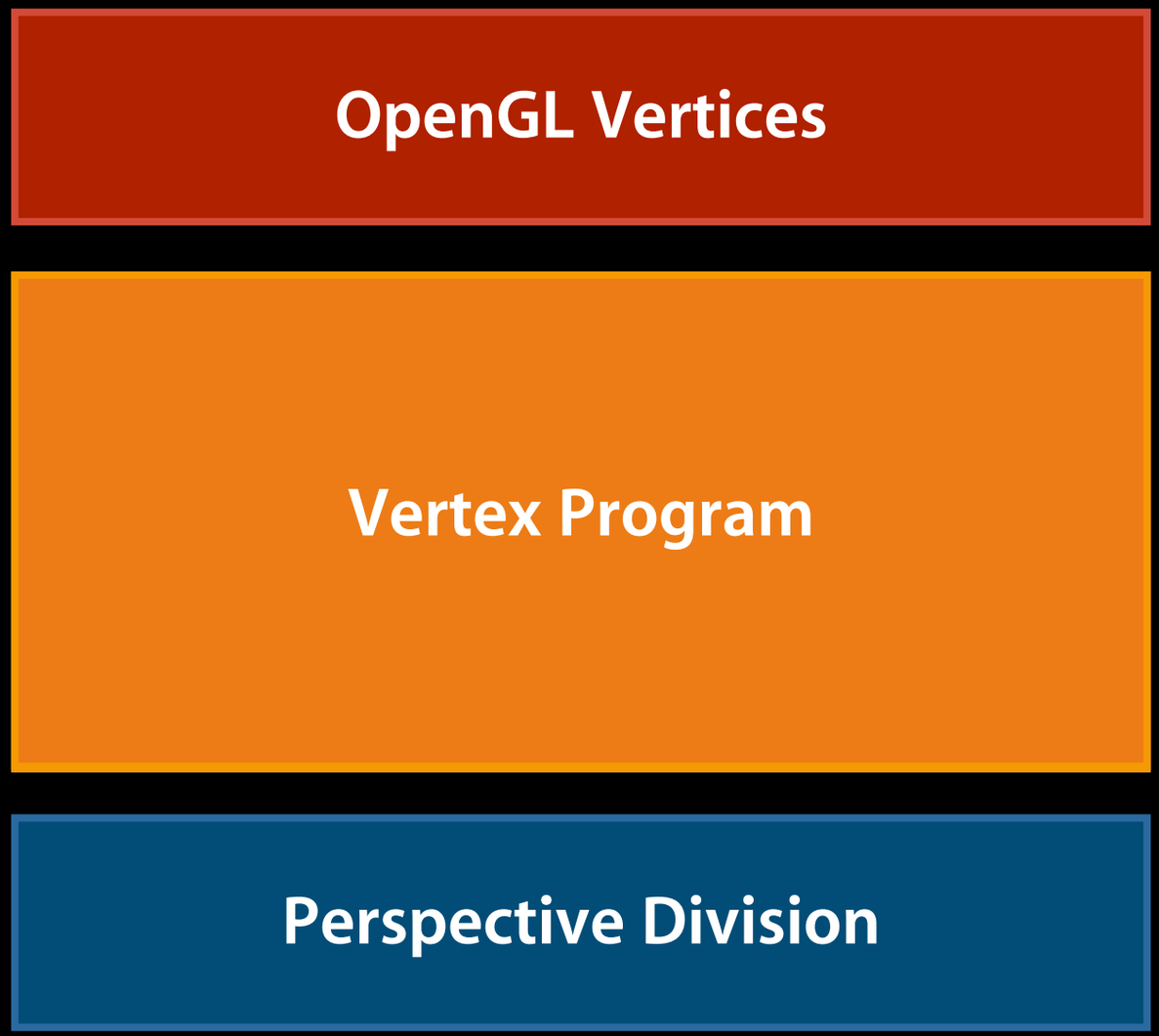
- Using fragment programs
- Application doesn't fit into the fixed pipeline
- Application is preprocessing vertex information
  - Surfaces
  - Vertex Tweening

Benefits of Vertex Programs

- Offloading functionality to GPU frees up CPU time
- Flexible per-vertex geometric operations
- Performance
- Lower memory usage

Vertex Programs In The OpenGL Pipeline

OpenGL Geometry Pipeline



What Changes?

- Vertex programs are now responsible for the model view transform, color material, lighting…
  - See ARB vertex program spec issues item 3
- When vertex programs are in use they must be disabled to enable the standard OpenGL geometry path

Computational Model

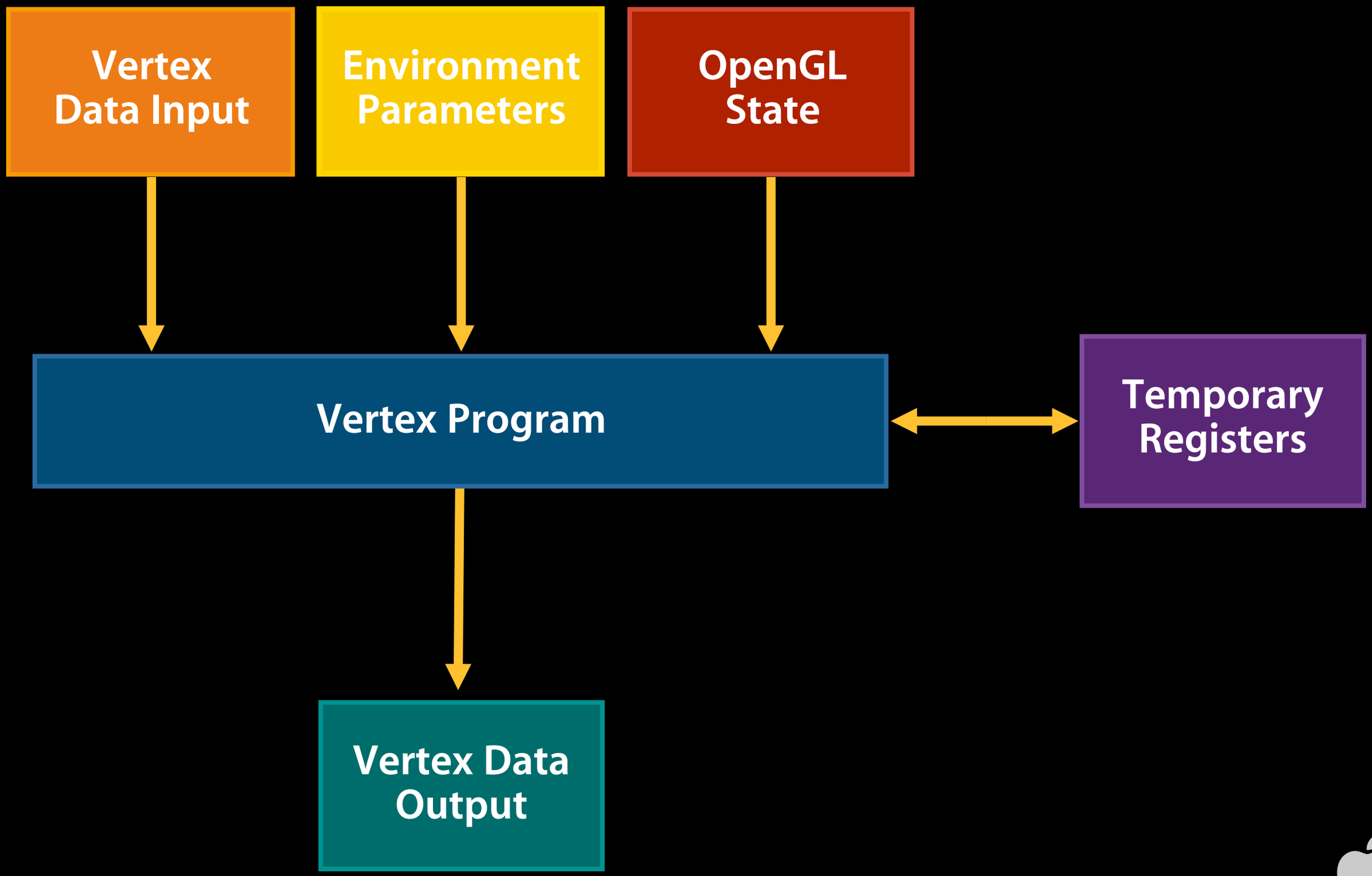
What Invokes a Vertex Program?

- The currently bound vertex program is executed when
  - A vertex is specified directly through the Vertex command
  - Indirectly through vertex arrays or evaluators
  - The current raster position is updated
- A program terminates after reaching the end of the program

Computation Model

- Independent execution on each vertex
- There can be (and likely are) multiple vertex units in GPU
- Temporary data is non-persistent

Program Input Output Model



Information Available for Programs

- Vertex Attributes
  - Standard attributes (Position, Color, Texture coordinates…)
  - Additional programmable attributes
- OpenGL Transform and Lighting state

Parameter Information

- Read only by program, constant data during execution
- Programmable from application
- Two types
  - Environment parameters are global across all vertex programs
  - Local parameters are for current vertex program

Instruction Set

- Limited instruction set with 27 instructions
- Focused on transform and lighting
- No loops or conditional branches
- Limited program length
  - Some instructions are macros, more than one instruction
  - Spec defines only a limited number of instructions that can be macros

Data Operands

- Scalar
  - Single floating point values
- SIMD
  - Four floating point values
  - Indexed using xyzw
- All Data is represented as SIMD float 4
  - Some operations have scalar destinations

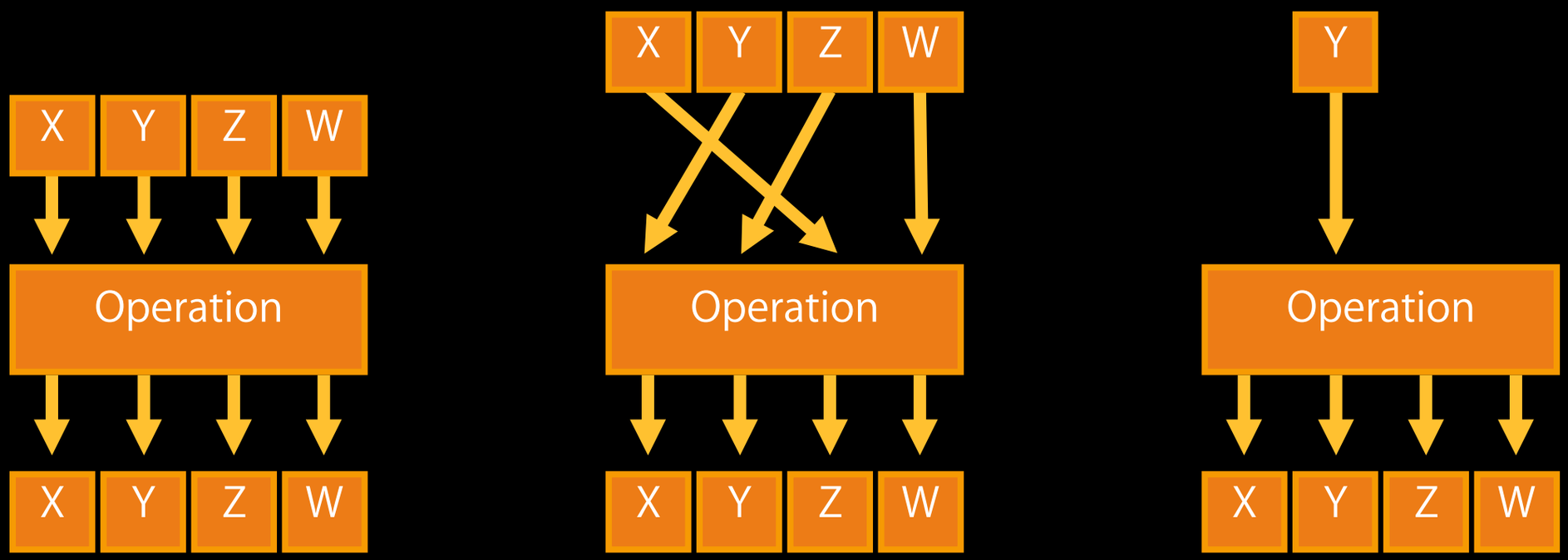
Data Component Selection

Operations can select source data on a component basis

Normal SIMD operation a = b

Swizzle SIMD operation a = b.yzxw

Scalar masked operation a = b.x



Standard Scalar / SIMD Instructions

- Add, Subtract, Multiply, Multiply add*
- Absolute value, Minimum, Maximum
- Dot product, Distance vector, Cross product
- Reciprocal

Math Instructions

- Exponent base 2
- Floor
- Fraction
- Log base 2

Complex Instructions

- Lighting
- Component selection
- Conditional set on compare
- Indirect register load

Vertex Program Constraints

- ARB vertex program specification
  - Minimum requirements
    - 96 environment program registers
    - 96 local program registers
    - 8 program matrices
    - 1 address register
    - 128 instructions
    - 12 temporary registers

What Does the Hardware Support?

- Use glGetProgramivARB and glGetIntegerv to query OpenGL what is supported
- See ARB vertex program specification on what enums to use to query this infromation

Syntax

Program Structure

- Must start with version signature
  - !!ARBvp1.0
- Must terminate with END statement
- Just about anything in between is ok
  - Loose on temporary and parameter definitions like C++

Syntax

- GPU independent assembly language
  - Not C
  - Not pure assembly
    - No fixed register assignments
- GPU independent
  - Run time compiled by driver to GPU
  - No fixed allocation of GPU registers

Syntax

- Parameters

  PARAM aConstParameter = {a, b, c, d};

  PARAM aEnvParameter = program.env[0];

  PARAM aLocalParameter = program.local[0];

- Temporary variables

  TEMP aTemp, anotherTemp;

- Unary, Binary and Ternary instructions

  ALU_OP dest, src;

  ALU_OP dest, src1, src2;

  ALU_OP dest, src1, src2, src3;

Instructions

Instruction	Inputs	Output	Description
ABS	V	V	absolute value
ADD	V, V	V	add
ARL	V	A	address register load
DP3	V, V	SSSS	3 component dot product
DP4	V, V	SSSS	4 component dot product
DPH	V, V	V	homogeneous dot product
DST	V, V	SSSS	distance vector
EXP2	V	V	exponential base 2
EXP	V	V	exponential base 2 (approximate)
FLR	V	V	floor
FRC	V	SSSS	fraction base 2
LG2	V	V	logarithm base 2
LIT	V	V	compute light coefficients
LOG	V	V	logarithm base 2 (approximate)
MAD	V,V	V	multiply and add

Instructions

Instruction	Inputs	Output	Description
MAX	V, V	V	maximum
MIN	V, V	V	minimum
MOV	V	A	move
MUL	V, V	V	multiply
POW	V	SSSS	exponentiate
RCP	V	SSSS	reciprocal
RSQ	V	SSSS	reciprocal square root
SGE	V, V	V	set on greater than or equal
SLT	V, V	V	set on less than
SUB	V, V	V	subtract
SWZ	V	V	extended swizzle
XPD	V	V	cross product

ARB Vertex Program Information

- OpenGL shader builder in developer tools
  - Good place to start
  - Active syntax checking
  - Displays results on sample objects
- Visit the Apple developer website for ARB_vertex_program spec

Loading Programs and Controlling the Environment

Loading Programs

- Loading the vertex program as a string
  - glProgramStringARB(GL_VERTEX_PROGRAM_ARB
  - Use GL_PROGRAM_FORMAT_ASCII_ARB for loading program from a C string
- Syntax error reporting
  - Test glGetError()
  - Use glGetString(GL_PROGRAM_ERROR_STRING_ARB) to return the syntax error

Controlling the Environment

- Loading parameters
  - Environment parameters
    - glProgramEnvParameter4fvARB(.....)
    - Loads 4 floating point values from a pointer to program.env[index]
  - Local parameters
    - glProgramLocalParameter4fvARB()
    - Loads 4 floating point values from a pointer to program.local[index]

Basic Vertex Program

Hello World Example

- Shader builder example passthrough.shdr
- MVP transform of input vertex
- Set vertex result

Demo

Lighting Vertex Program

Lighting Example

- Shader builder example AdvancedLightingModel.shdr
- Model view transform of input vertex
- Perform lighting computations
  - Compute surface normal
  - Compute lighting
- Set vertex result

Demo

Shader Builder Examples

- Shader builder is a great place to start
- No framework construction required
- Provides active syntax checking
- Online instruction information

Program Possibilities

- Instruction set is small, code space is smaller, and the number of temporary registers is even smaller
- But you would be amazed what can be done
  - Surfaces
  - Displacements
  - Advanced lighting effects
  - Animation
  - Visual computation

Programming Tips

- Use vertex arrays
- Use vertex array ranges

Programming Tips

- Use component selection to save parameter place
  - Example

```
PARAM one = {1.0, 1.0, 1.0, 1.0};
PARAM two = {2.0, 2.0, 2.0, 2.0};
```

  - Try

```
PARAM oneTwo = {1.0, 2.0, 1.0, 0.5};
```

- Use compound parameters

```
PARAM fact_6 = {720, -720, 0.00138889, -0.00138889};
```

Programming Tips

- Scalar divide / square root
  - Only one component can be selected

```
RCP aTemp, aTemp.y
```

  - Use a select parameter and MAD to merge back together

```
PARAM select = {1,0,0,0}
RCP aTemp0, aTemp.x
RCP aTemp1, aTemp.y
RCP aTemp2, aTemp.z
MAD aTemp, aTemp0, select.xyyy, zero
MAD aTemp, aTemp1, select.yxyy, aTemp
MAD aTemp, aTemp2, select.yyxy, aTemp
```

Programming Tips

- Conditional selection

  if (a >= b) then c = a

  else c = b

```
SGE temp0, a, b;
SLT temp1, b, a;
MUL temp0, a, temp0;
MAD temp1, b, temp1, temp0;
```

Programs for Surfaces

Vertex Programs for Surfaces

- Use a UV mesh for surface interpolation by vertex program
- Vertex programs can be used for
  - Quadradic surfaces (spheres, cylinders…)
    - $f(\theta, \Phi) = r$
  - Implicit surfaces
    - $f(x,y,z) = 0$
  - Parametric surfaces (Beizer, B-Spline, NURBS)
    - $x(u,v)$, $y(u,v)$, $z(u,v)$

Vertex Programs for Surfaces

- What is a UV mesh?
  - A bounded 2D mesh for U and V
    - Example: parametric surfaces bound from 0 to 1
- Why use vertex programs for surfaces?
  - Single UV mesh required for all surfaces
  - Animate surfaces using control parameters
  - Offloading work from CPU to GPU

Implicit Surface Vertex Program

Cool Math Thingy

- Surface is defined by

  x = sin(u)

  y = sin(v)

  z = sin((dd - aa * u - bb * v) / cc)

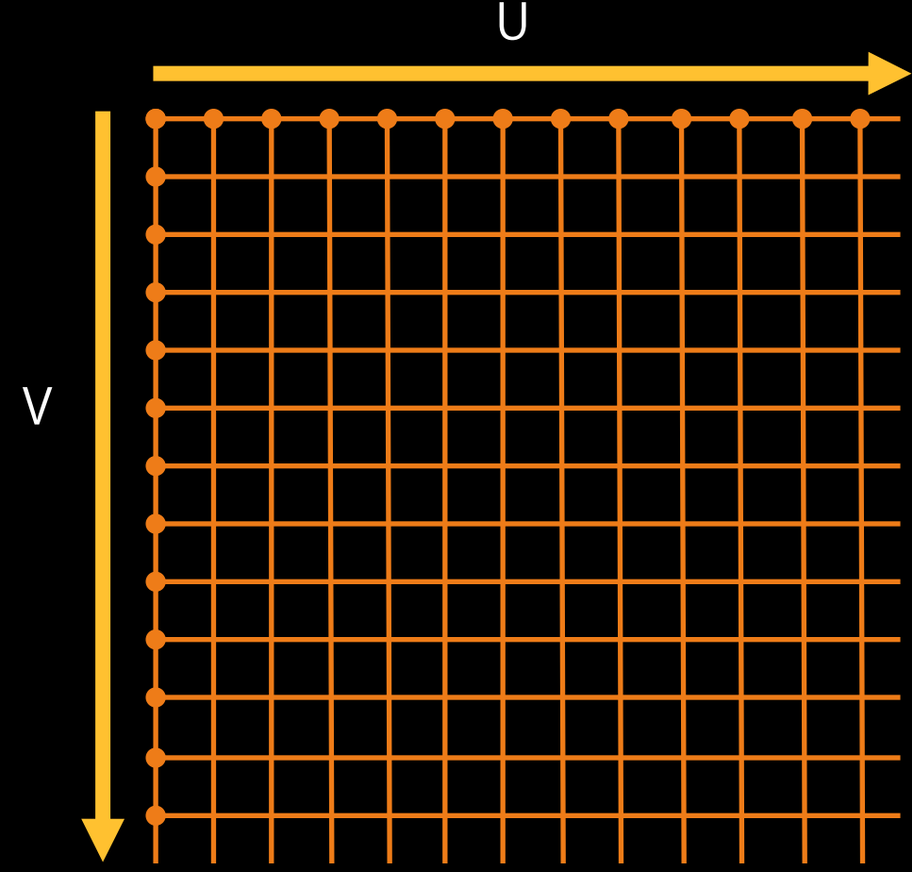
Building a UV Mesh

- Build UV mesh as vertex array
  - UV mesh is bounded by -π and π for U and V
  - Fill in position with UV mesh

    x = u

    y = v

    z = 0



Drawing a UV mesh

- Load and enable vertex program
  - glProgramStringARB(.....)
- Load shape control parameters (a, b, c, d)
  - glProgramLocalParameter4fvARB(.....)
- Submit quad strip vertex array for drawing
  - glDrawElements for a vertex array

Demo

Implicit Surface Demo

- Vertex program computed object shape and color
- Control values a, b, c, d were uploaded as parameters
- There's no Sine(u,v,…) function
  - Sine was computed using a Maclaurin power series

    sin(x) = x - 1/6 * x^3 + 1/120 * x^5 - 1/5040 x^7 + 1/362880 * x^9.....
  - Vertex program computed all 3 sine values at once using SIMD math

Implicit Surface Program

```
# Input parameter from application
PARAM          ABCD             = program.local[0];

# Tempory variables
TEMP  tPos, aPos, temp0, temp1;


# Surface is described by
#   x = sin(u);  y = sin(v);  z = sin((dd - aa * u - bb * v) / cc);

MUL    temp0,  ABCD.xxxx,      vPos.xxxx;   # aa * u
SUB    temp0,  ABCD.wwww, temp0.xxxx;   # dd - (aa * u)
MUL    temp1,  ABCD.yyyy,     vPos.yyyy;    # bb * v
SUB    temp0,  temp0.xxxx,    temp1.xxxx;  # dd - (aa * u) - (bb * v)
RCP    temp1,  ABCD.z;
MUL    temp0,  temp0,  temp1;
SWZ    temp0,  temp0,  0,0,z,0;
SWZ    aPos,     vPos,      x,y,0,1;
ADD    aPos,     aPos,      temp0;                    # aPos = u, v, ((dd - (aa * u) - (bb * v)) / cc)



# sin(x) = x - 1/6 * x^3 + 1/120 * x^5 - 1/5040 x^7 + 1/362880 * x^9.....
MUL    temp0, aPos, aPos;                            # temp0 = xyz^2

MUL    temp1, temp0, aPos;                  # temp1 = xyz^3                n=1
MAD    tPos, temp1, fact_3.wwww,  aPos;    # temp0 = x - 1/6 * x^3
```

Implicit Surface Program

```
Do a few more terms.....
MUL    temp1, temp1, temp0;                        # temp1 = xyz^19       n=9
MAD    tPos, temp1, fact_19.wwww, tPos;    # temp0 = x - 1/6 * x^3 + 1/120 * x^5 - 1/5040 * x^7.....

MUL    temp1, temp1, temp0;                        # temp1 = xyz^21       n=10
MAD    tPos, temp1, fact_21.zzzz, tPos;            # temp0 = x - 1/6 * x^3 + 1/120 * x^5 .... = sin(x)



# Set the w to 1
SWZ    temp0, tPos, x,y,z,1;

# Transform the vertex by the modelview/projection matrix
DP4 result.position.x,  state.matrix.mvp.row[0],  temp0;
DP4 result.position.y,  state.matrix.mvp.row[1],  temp0;
DP4 result.position.z,  state.matrix.mvp.row[2],  temp0;
DP4 result.position.w, state.matrix.mvp.row[3],  temp0;



# Set Texture coordinate
SWZ  temp0, EQ0123, x,0,0,0;
ADD  temp0, temp0, vTexcoord;
MOV result.texcoord, temp0;

END
```

Get Your Feet Wet

- Try doing waves
  - Use muliple cosine emitters at different locations

    $z = \sum$ Amplitude * cos(time * freq) * exp(decay/distance)

    - See demo in OpenGL optimizations

NURBS Vertex Program

Why NURBS

- Why use vertex programs for NURBS?
  - Offloading work to GPU
  - Multiple patches can be stored using control meshes and a UV mesh
  - Automatic level of detail is possible using on the fly UV mesh generation

NURBS

- Parametric surface using Non-uniform rational B-Splines
  - UV mesh is bound between 0 and 1
  - Surface is defined by interpolating control points
- Search web for introduction to NURBS
  - *Introduction to NURBS* on the web
  - *An Introduction to NURBS* by David F. Rogers

NURBS Using Vertex Programs

- Outline of how to draw NURBS using vertex program
  - B-Spline basis functions are computed on host CPU for UV mesh
  - Load control points as local program parameters
  - Position / Normal / Texture coordinates are interpolated from control points by vertex program

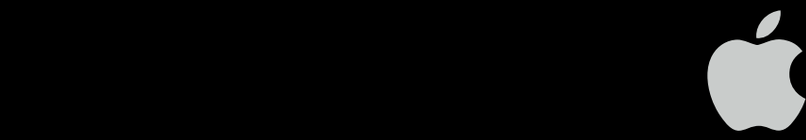
NURBS Using Vertex Programs

- Example
  - Uses a 4x4 control mesh
  - Position information interpolated by vertex program
- Implementation
  - Each UV vertex has 16 basis functions

$$P(u,v) = \sum C(n)\, M(u, k)\, N(v, k)$$

n = 0->16, k = 3

  - Post multiply M(u, k) and N(v, k) on vertex program to lower storage cost from 48 floats to 8 floats



Demo

NURBS Demo

- Mesh was 100 by 100
  - 10,000 point mesh
  - 960,000 FLOPS that the host didn’t do
  - Full screen application at 75 fps > 1 GFLOP

NURBS Program

```
!!ARBvp1.0

ATTRIB     uBasis      = vertex.position;
ATTRIB     vBasis      = vertex.color;

#Position control points
PARAM      P00         = program.local[0];
PARAM      P01         = program.local[1];
PARAM      P02         = program.local[2];
PARAM      P03         = program.local[3];
PARAM      P10         = program.local[4];
.... removed for slides  ....

# Row 0
MUL  M0,  uBasis.xxxx,     vBasis.xxxx;        # Create 4 column basis functions from K[0]J[0..3]
MUL  M1,  uBasis.yyyy,     vBasis.xxxx;
MUL  M2,  uBasis.zzzz,     vBasis.xxxx;
MUL  M3,  uBasis.wwww,     vBasis.xxxx;

MUL  P, M0, P00;                          # Start evaluating P from control points and basis functions
MAD  P, M1, P01, P;                       # Multiply and accumulate as we go
MAD  P, M2, P02, P;
MAD  P, M3, P03, P;
```

NURBS Program

```
# Rows 1and 2 skipped for presentation
# Row 3
MUL  M0, uBasis.xxxx, vBasis.wwww;
MUL  M1, uBasis.yyyy, vBasis.wwww;
MUL  M2, uBasis.zzzz, vBasis.wwww;
MUL  M3, uBasis.wwww, vBasis.wwww;

MAD  P, M0, P30, P;
MAD  P, M1, P31, P;
MAD  P, M2, P32, P;
MAD  P, M3, P33, P;

# Transform the vertex by the modelview/projection matrix
SWZ  P, P, x,y,z,1;
DP4    result.position.x, state.matrix.mvp.row[0], P;
DP4    result.position.y,  state.matrix.mvp.row[1], P;
DP4    result.position.z,  state.matrix.mvp.row[2], P;
DP4    result.position.w, state.matrix.mvp.row[3], P;

# Pass on the texture information
MOV    result.texcoord, vertex.texcoord;

END
```

NURBS Demo

- Extending the example
  - Import position, normal and texture control meshes from modeling program
  - Vertex programs for 3x3, 3x4, 3x5, 4x3, 4x4, 5x3 control meshes
  - Subdivision surfaces
  - Exact solution in Siggraph 98

Wrap Up

Vertex Programs

- Powerful tool for shading and lighting
  - Custom lighting models
- Additional complexity in geometry to be defined
  - UV meshes can define most surfaces
  - A UV mesh can be loaded in VRAM to reduce AGP traffic
- Lowers CPU load for applications
- Performance
  - Up to 100 times faster than non-optimized cases

Getting Started

- Start with shader builder
  - Simple transforms
  - Lighting models
- Move onto simple UV meshes
  - Implicit surface generation
- Then explore
  - B-Splines, NURBS, Subdivision surfaces
  - Lighting models

More Information

- ARB Vertex program spec
  - Long and very detailed, but precise
- WWDC examples will be posted on Apple OpenGL website
- Mac-OpenGL mailing list

Roadmap

200 Graphics and Imaging Overview
Latest developments in Mac OS X graphics APIs.
Presidio
Tue, 10:30am

201 Mac OS X OpenGL in Depth
Introduction to OpenGL on Mac OS X.
Nob Hill
Tue, 2:00pm

202 Technology Magic of Keynote
How Keynote uses Mac OS X graphics technologies.
Nob Hill
Tue, 3:30pm

204 Image Capture Update
Integrating Image Capture with your product.
Marina
Wed, 9:00am

Roadmap

205 Vertex Programming with OpenGL Super-fast geometry transformations using the GPU.	**Marina** **Wed, 3:30pm**
206 ColorSync Update Latest developments in Mac OS X color management.	**Mission** **Wed, 5:00pm**
207 Quartz 2D in Depth Harnessing the power of Quartz 2D in your product.	**Presidio** **Thu, 9:00am**
208 Fragment Programming with OpenGL Creating incredible per-pixel effects on Mac OS X.	**Pacific Heights** **Thu, 10:30am**

Roadmap

209 OpenGL Optimizations Obtaining maximum 3D performance on Mac OS X.	**Marina** **Thu, 2:00pm**
210 Mac OS X Printing Update Latest developments in the Mac OS X printing system.	**Mission** **Thu, 3:30pm**
211 Introduction to Quartz Services Capturing & configuring displays on Mac OS X.	**Pacific Heights** **Fri, 9:00am**
212 Cutting Edge OpenGL Techniques Expert recipes for stunning 3D effects.	**Russian Hill** **Fri, 2:00pm**

Roadmap

421 Mac OS X Accessibility
Latest developments in Mac OS X accessibility.

Presidio
Fri, 10:30am

FF018 Graphics & Imaging Feedback
Share your thoughts on graphics & imaging technologies.

North Beach
Fri, 5:00pm

Who to Contact

Michael K. Larson
Graphics Guy
mklarson@apple.com

Geoff Stahl
The Other Graphics Guy
gstahl@apple.com

http://developer.apple.com/wwdc2003/urls.html

Addtional Information

Reference Library

Documentation

ARB Vertex Programing Spec
oss.sgi.com/projects/ogl-sample/registry/ARB/vertex_program.txt

Introduction to NURBS
libnurbs.sourcefoge.net/nurbsintro.pdf

An Introduction to NURBS
David F. Rogers

Apple OpenGL Developer
developer.apple.com/opengl

Q&A

Michael K. Larson
Graphics Guy
mklarson@apple.com

http://developer.apple.com/wwdc2003/urls.html

WWDC 2003

Worldwide Developers Conference